OOCP C++

C++ is an object-oriented programming language.

Developed by Bjarne Stroustrup at AT&T Bell Laboratories, USA in the early 1980’s.

1997 November ANSI/ISO standards committee standardized C++.

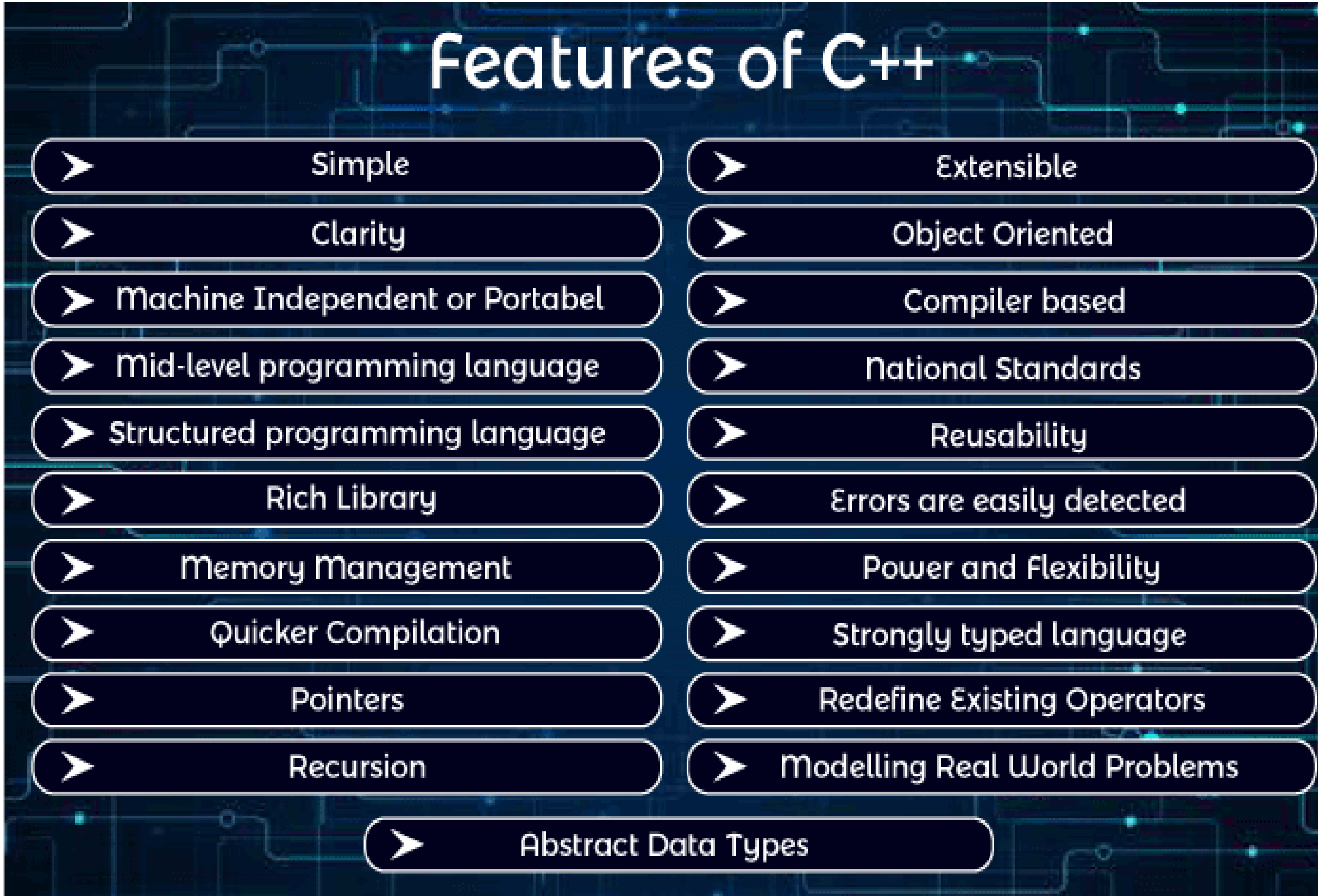
For developing editors, compilers, databases, communication systems, etc.

C++ works by giving (separate) instructions to the computer.

These instructions can be written as functions. The primary function used in C++ is called **main**.

The body of a function starts with an opening curly bracket "{" and closes with a closing curly bracket "}".

C++ is a widely used programming language.



Features of C++

- Simple
- Clarity
- Machine Independent or Portabel
- Mid-level programming language
- Structured programming language
- Rich Library
- Memory Management
- Quicker Compilation
- Pointers
- Recursion
- Extensible
- Object Oriented
- Compiler based
- National Standards
- Reusability
- Errors are easily detected
- Power and Flexibility
- Strongly typed language
- Redefine Existing Operators
- Modelling Real World Problems
- Abstract Data Types

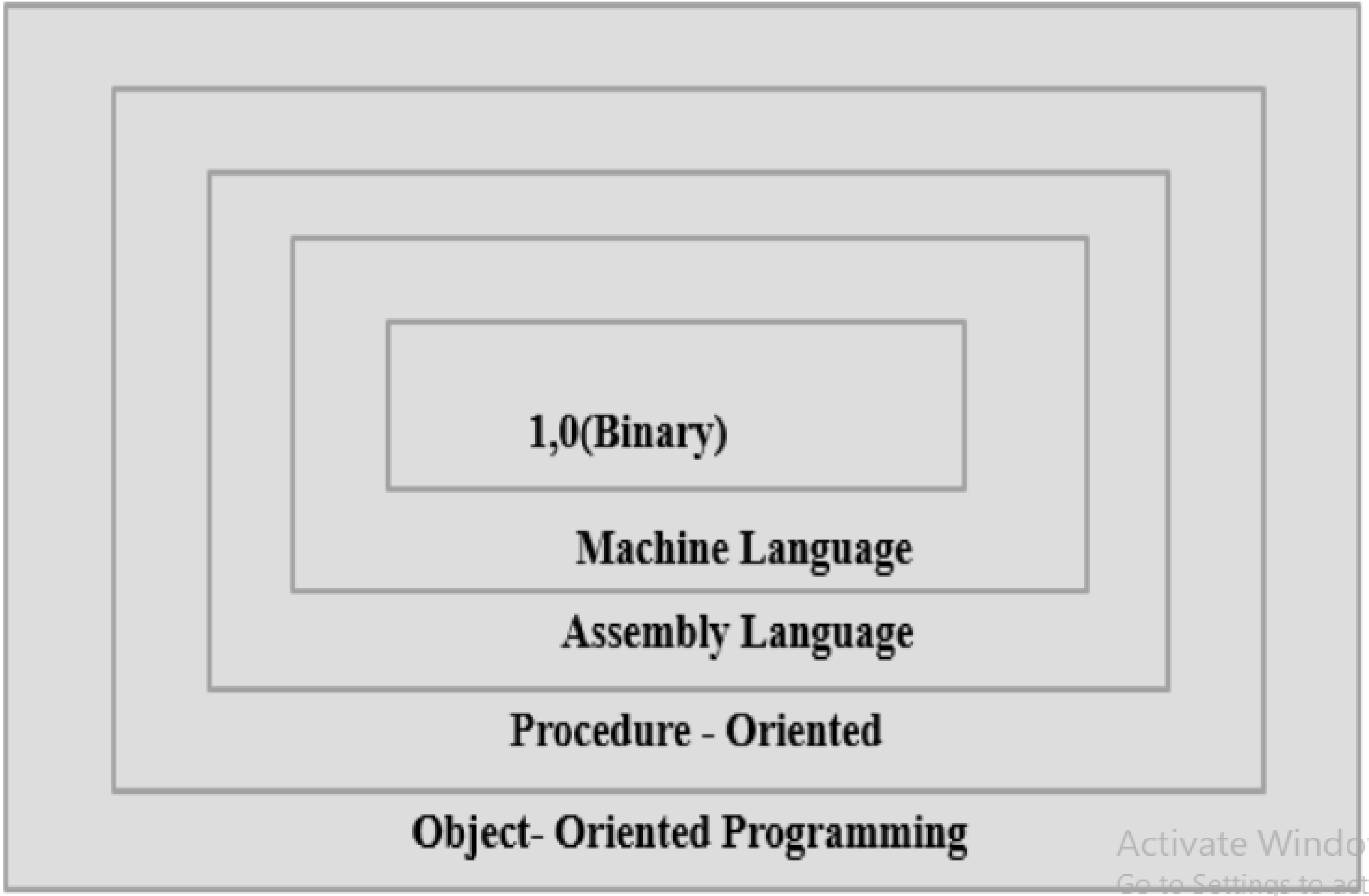
Software technology is dynamic as continuous new approach to software design and development.

Software product should be evaluated carefully for their quality before they are delivered and implemented.

Some quality issues that considered as……

1. Correctness
2. Security ⍰
3. Maintainability
4. Integrity
5. Reusability 6. User Friendliness 7 Portability

- **Layers of computer software**



1,0(Binary)

Machine Language

Assembly Language

Procedure - Oriented

Object- Oriented Programming

Activate Windows
Go to Settings to activ

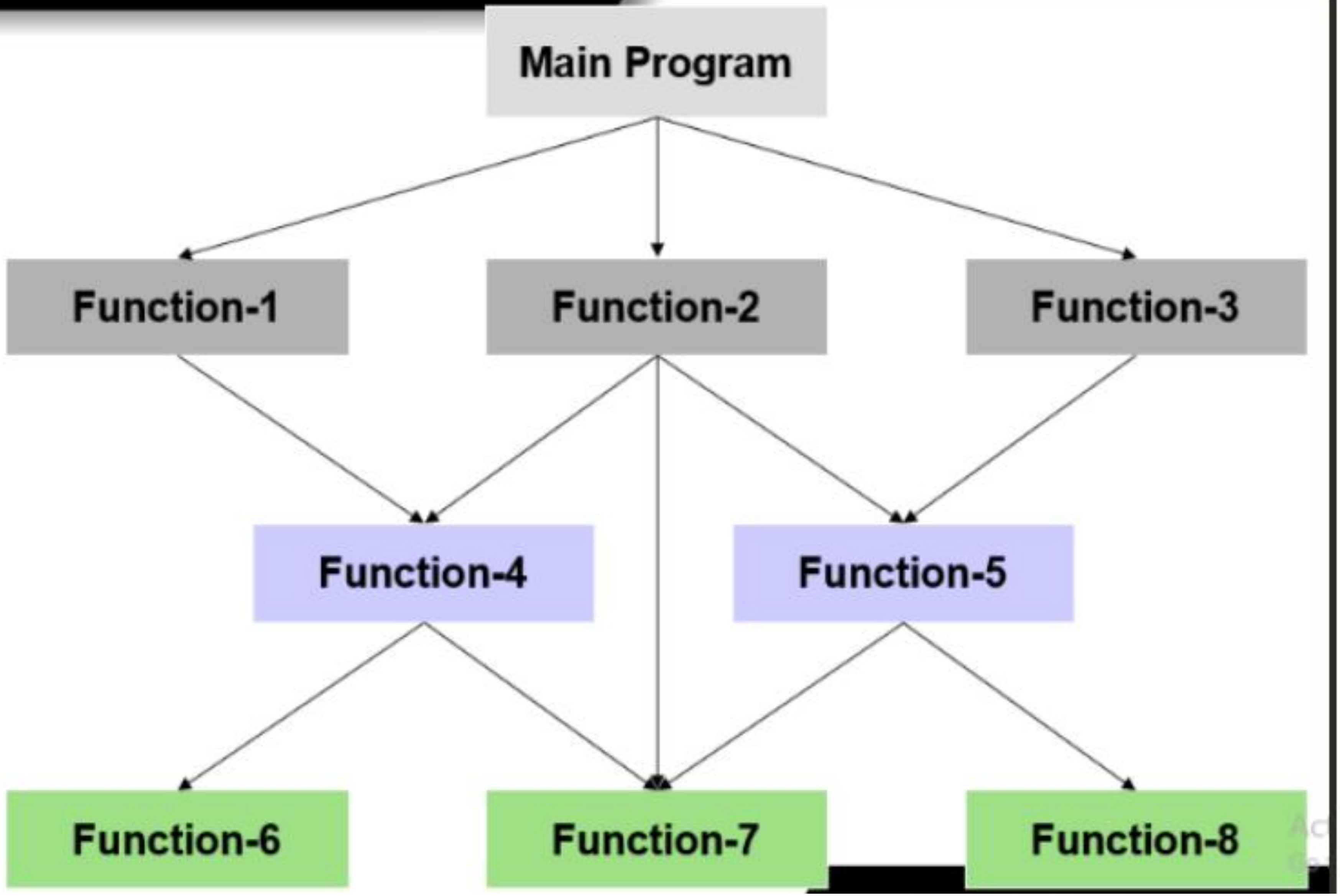
- S/w evolution has distinct phases or “layers” or growth. Each layer has improvement over previous one. Each layer work as functional.
- Modular Programming, top-down programming, bottom-up programming and structured programming are different techniques of programming.
- Structured Programming was powerful tool that enable programmers to write moderately complex programs fairly easily.
- OOP is new way of organizing and developing programs.

- Traditional procedural language, such as assembly language or a high-level like COBOL, FORTRAN, C, etc.

- The problem is viewed as a sequence of things to be done.

- The primary focus is on functions.

- Procedure-oriented programming basically consists of writing a list of instructions for the computer to follow and organizing these instructions into groups known as functions.



Main Program

Function-1

Function-2

Function-3

Function-4

Function-5

Function-6

Function-7

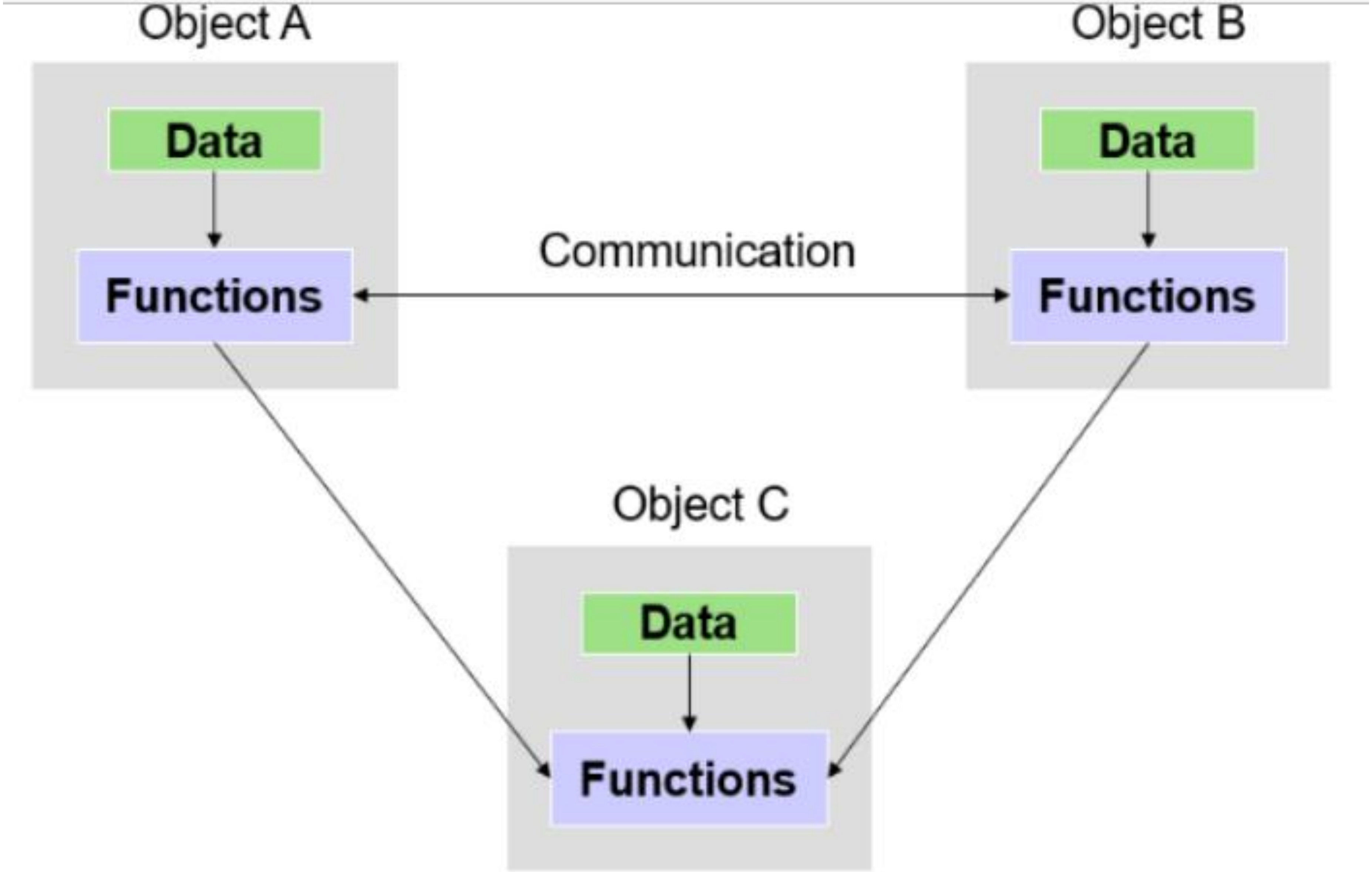
Function-8

- **Object-Oriented Programming (OOP)** is the term used to describe a programming approach based on **objects** and **classes**. The object-oriented paradigm allows us to organise software as a collection of objects that consist of both data and behaviour.

- The data of an object can be accessed only by the functions associated with that object.

- Functions of one object can access the functions of another objects.

- Example: Class Fruit, that have two object mango(sweet and yellow) and apple(sweet and red color). In both have some functions are same and some not same.



Object A

Data

Functions

Communication

Object B

Data

Functions

Object C

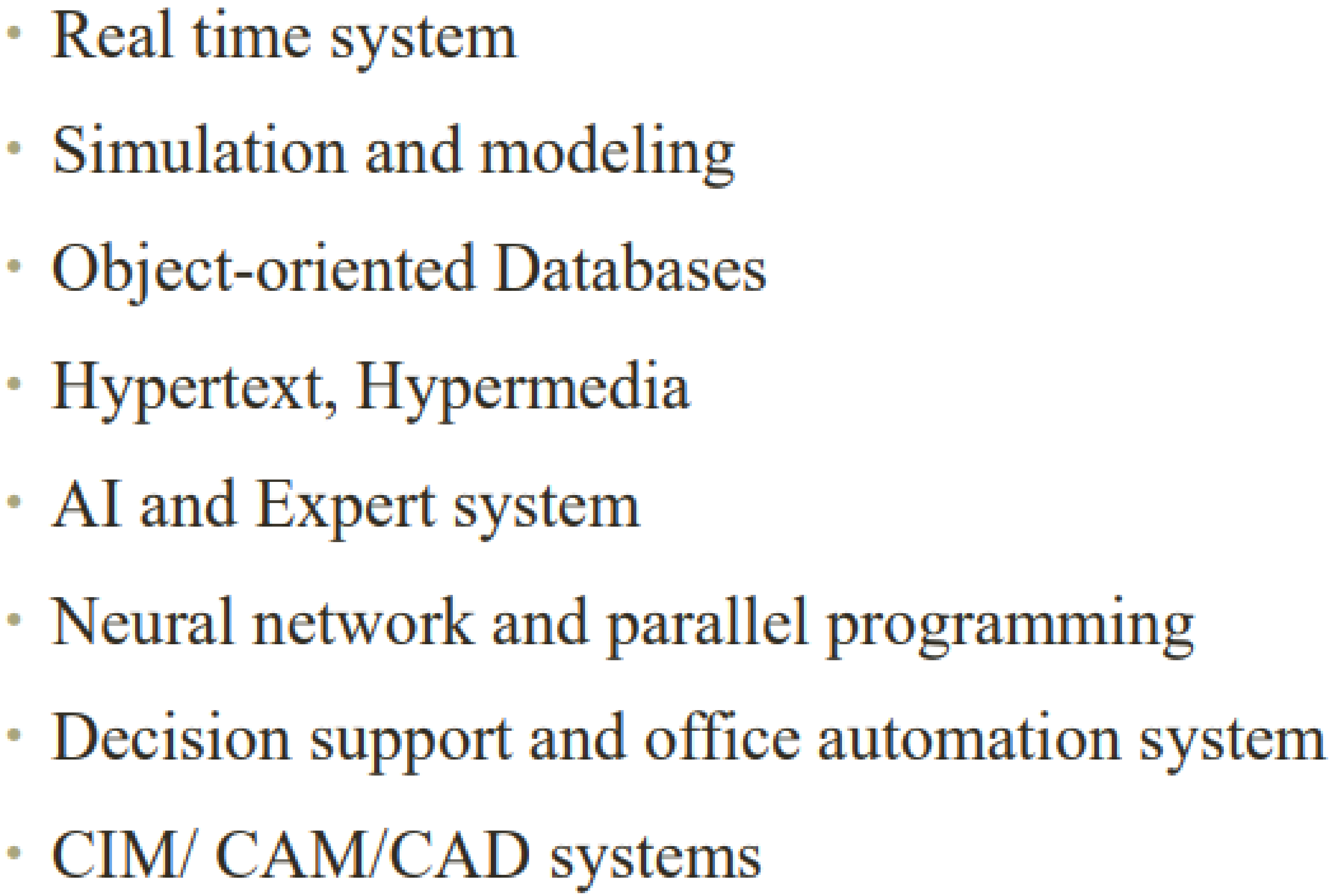
Data

Functions

POP	OOP
Emphasis is doing on things not on data, so it is procedure oriented.	Emphasis is on data, so it is object oriented.
Main focus is on function and procedures that operate on data.	Main focus is data that is being operated.
Top-Down approach in program design	Bottom-up approach in program design.
Larger program is divide in smaller parts known as function.	Large program is divided into classes and objects.
Most of function share global data.	Data is tied together with function in data structure.

Data moves openly from one function to another function.	Data is hidden and can't be accessed external events.
Adding of function and data is difficult.	Adding of function and data is easy.
We can't declare namespace directly.	We can use name space directly; eg. Using namespace std;
Inheritance, polymorphism, abstraction, access specified are not supported.	Inheritance, polymorphism, abstraction, access specified are supported.
Eg. FORTRAN, C, Pascal etc…	Eg. C++, Java, C# etc…

Applications of OOP



- Real time system
- Simulation and modeling
- Object-oriented Databases
- Hypertext, Hypermedia
- AI and Expert system
- Neural network and parallel programming
- Decision support and office automation system
- CIM/ CAM/CAD systems

A Simple C++ Program

```
#include <iostream> //include header file
using namespace std;
int main()
{
        cout << "Hello World"; // C++ statement
        return 0;
}
```

- `iostream` is just like we include `stdio.h` in c program.
- It contains declarations for the identifier `cout` and the insertion operator `<<`.
- `iostream` should be included at the beginning of all programs that use input/output statements.

A Simple C++ Program (Cont…)

```cpp
#include <iostream> //include header file
using namespace std;
int main()
{
      cout << "Hello World"; // C++ statement
      return 0;
}
```

- A namespace is a declarative region.
- A **namespace** is a part of the program in which certain names are recognized; outside of the namespace they’re unknown.
- namespace defines a scope for the identifies that are used in a program.
- **using** and **namespace** are the keywords of C++.

A Simple C++ Program (Cont…)

```
#include <iostream> //include header file
using namespace std;
int main()
{
     cout << "Hello World"; // C++ statement
     return 0;
}
```

- **std** is the namespace where ANSI C++ standard class libraries are defined.
- Various program components such as **cout, cin, endl** are defined within **std** namespace.
- If we don’t use the **using** directive at top, we have to add the **std** followed by **::** in the program before identifier.

```
std::cout << “Hello World”;
```

A Simple C++ Program (Cont…)

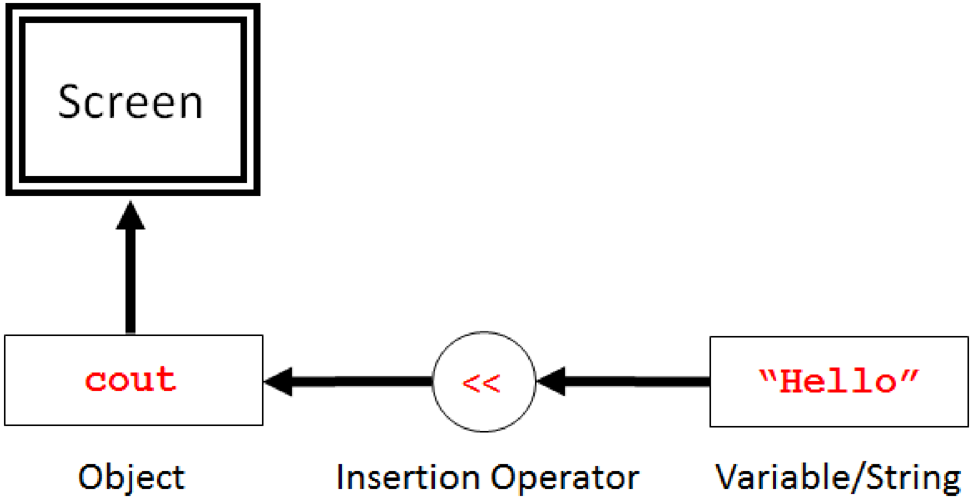
```
#include <iostream> //include header file
using namespace std;
int main()
{
        cout << "Hello World"; // C++ statement
        return 0;
}
```

- In C++, **`main()`** returns an integer type value.
- Therefore, every **`main()`** in C++ should end with a **`return 0;`** statement; otherwise error will occur.
- The return value from the **`main()`** function is used by the runtime library as the **`exit code`** for the process.

Insertion Operator <<

```
cout << "Hello World";
```

- The operator **<<** is called the insertion operator.
- It inserts the contents of the variable on its right to the object on its left.
- The identifier **cout** is a predefined object that represents standard output stream in C++.
- Here, Screen represents the output. We can also redirect the output to other output devices.
- The operator **<<** is used as bitwise left shift operator also.



Output Using Insertion Operator

Program: Basic C++ program

Write a C++ Program to print following

```
Name: xyz
City: Rajkot
Country: India
```

Program: Basic C++ program

```
#include <iostream>
using namespace std;
int main()
{
      cout << "Name: xyz";
      cout << "City: Rajkot";
      cout << "Country: India";
      return 0;
}
```

Output

```
Name: DarshanCity: RajkotCountry: India
```

Program: Basic C++ program(Cont…)

```
#include <iostream>
using namespace std;
int main()
{
 cout << "Name: xyz\n";
 cout << "City: Rajkot\n";
 cout << "Country: India";
 return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
 cout << "Name: xyz"<<endl;
 cout << "City: Rajkot"<<endl;
 cout << "Country: India"<<endl;
 return 0;
}
```

Output

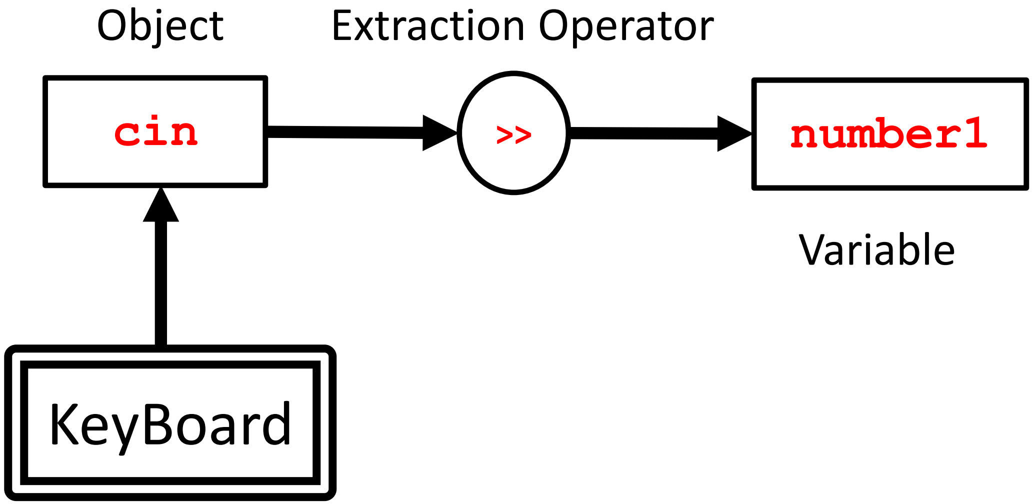
```
Name: xyz
City: Rajkot
Country: India
```

- The **endl** manipulator and **\n** has same effect. Both inserts new line to output.
- But, difference is **endl** immediate flush to the output while **\n** do not.

Extraction Operator >>

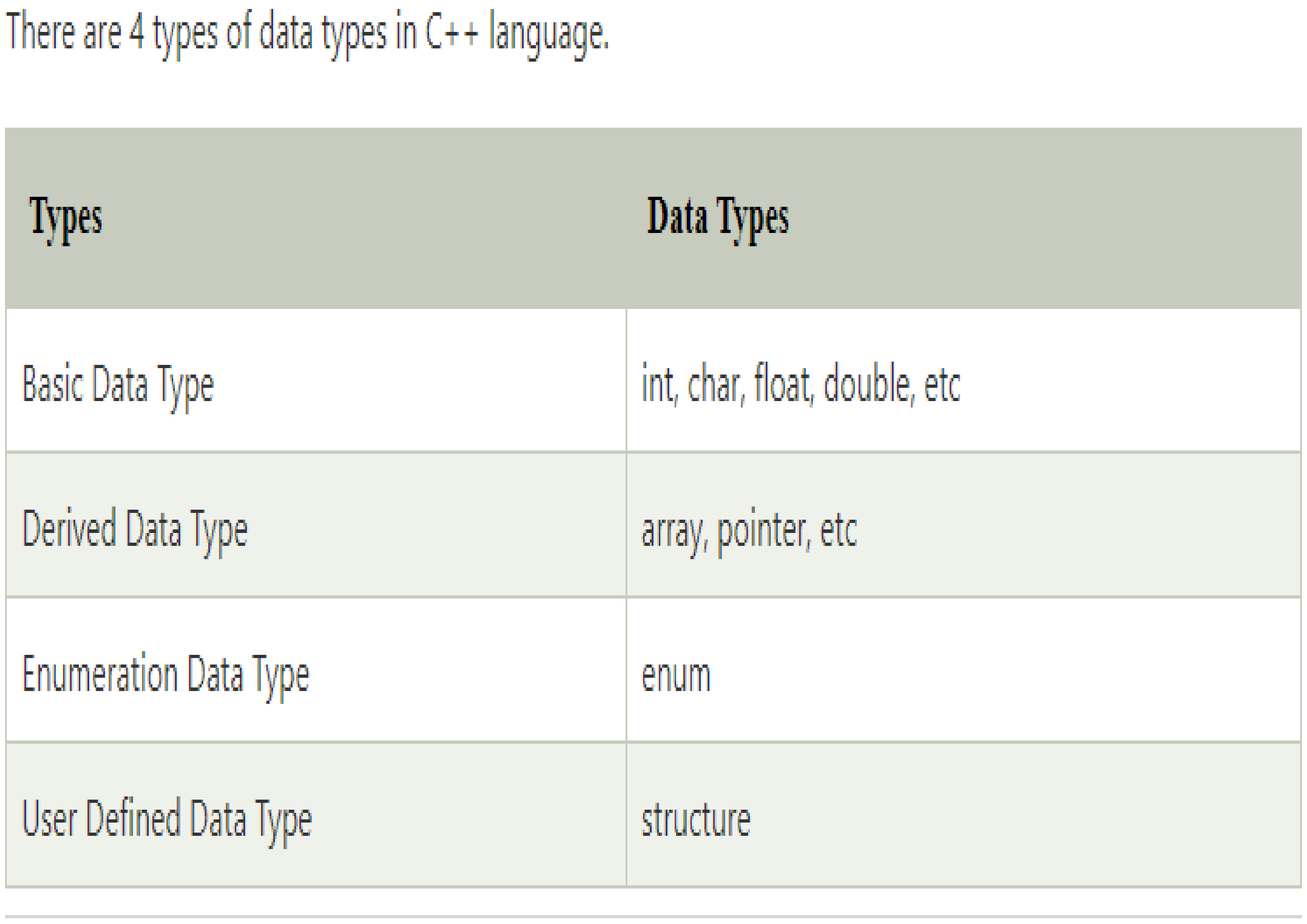
```
cin >> number1;
```

- The operator **>>** is called the extraction operator.
- It extracts (or takes) the value from keyboard and assigns it to the variable on its right.
- The identifier **`cin`** is a predefined object that represents standard input stream in C++.
- Here, standard input stream represents the Keyboard.
- The operator **>>** is used as bitwise right shift operator also.



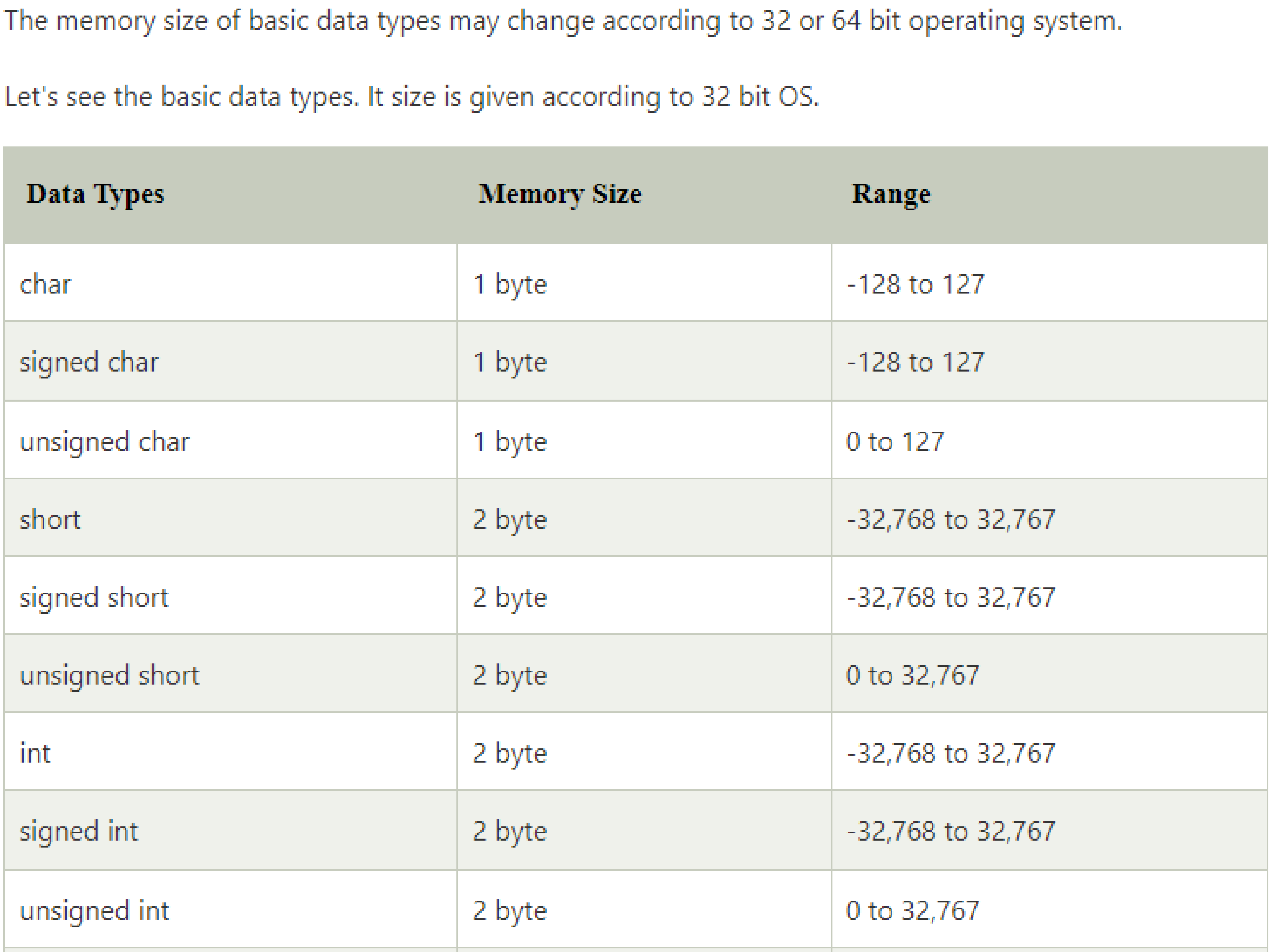
Program: Basic C++ program

```
#include<iostream>
using namespace std;
int main()
{
  int number1,number2;

  cout<<"Enter First Number: ";
  cin>>number1;                 //accept first number

  cout<<"Enter Second Number: ";
  cin>>number2;                 //accept first number

  cout<<"Addition : ";
  cout<<number1+number2;      //Display Addition
  return 0;
}
```



There are 4 types of data types in C++ language.

Types	Data Types
Basic Data Type	int, char, float, double, etc
Derived Data Type	array, pointer, etc
Enumeration Data Type	enum
User Defined Data Type	structure



The memory size of basic data types may change according to 32 or 64 bit operating system.

Let's see the basic data types. It size is given according to 32 bit OS.

Data Types	Memory Size	Range
char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 127
short	2 byte	-32,768 to 32,767
signed short	2 byte	-32,768 to 32,767
unsigned short	2 byte	0 to 32,767
int	2 byte	-32,768 to 32,767
signed int	2 byte	-32,768 to 32,767
unsigned int	2 byte	0 to 32,767

C++ Tokens

C++ Tokens

- The smallest individual unit of a program is known as **token**.
- C++ has the following tokens:
  - Keywords
  - Identifiers
  - Constants
  - Strings
  - Special Symbols
  - Operators

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World";
    return 0;
}
```

Keywords and Identifier

- C++ reserves a set of 84 words for its own use.
- These words are called **keywords** (or reserved words), and each of these keywords has a special meaning within the C++ language.
- **Identifiers** are names that are given to various user defined program elements, such as variable, function and arrays.
- Some of Predefined **identifiers** are cout, cin, main

❑ We cannot use Keyword as user defined identifier.

Keywords in C++

asm

auto

break

case

catch

char

class

const

continue

default

delete

do

double

else

enum

extern

float

for

friend

goto

if

inline

int

long

new

operator

private

protected

public

register

return

short

signed

sizeof

static

struct

switch

template

this

throw

try

typeof

union

unsigned

virtual

void

volatile

while

Rules for naming identifiers in C++

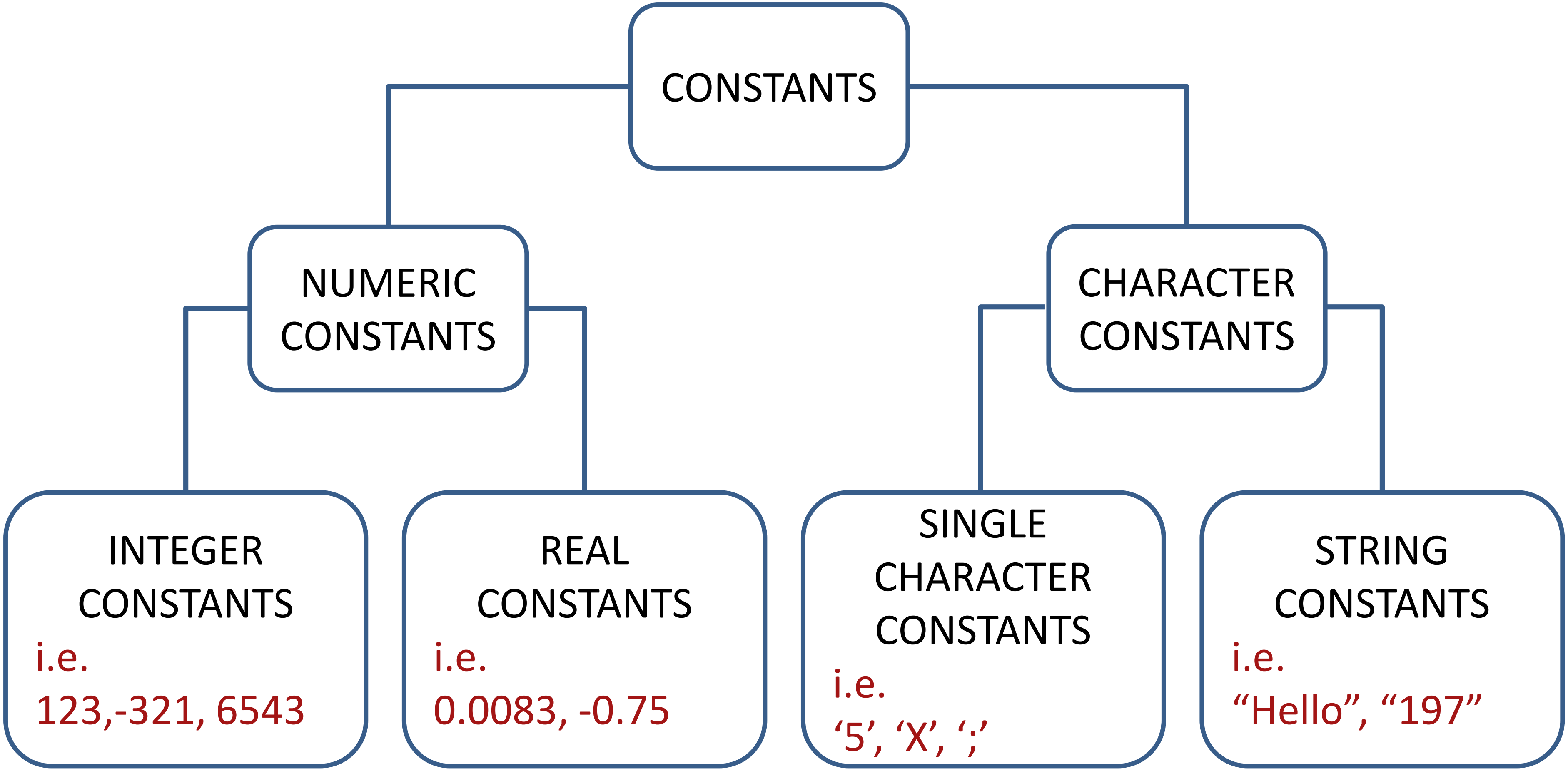
1. First Character must be an **alphabet or underscore.**
2. It can contain **only letters**(a..z A..Z), **digits**(0 to 9) or **underscore**(_).
3. Identifier name cannot be **keyword.**
4. Only first **31 characters** are significant.

Valid, Invalid Identifiers

1) Darshan	Valid	12) xyz123	Valid
2) A	Valid	13) part#2	Invalid
3) Age	Valid	14) "char"	Invalid
4) void	Reserved word	15) #include	Invalid
5) MAX-ENTRIES	Invalid	16) This_is_a_	Valid
6) double	Reserved word	17) _xyz	Valid
7) time	Valid	18) 9xyz	Invalid
8) G	Valid	19) main	Standard identifier
9) Sue's	Invalid	20) mutable	Reserved word
10) return	Reserved word	21) double	Reserved word
11) cout	Standard identifier	22) max?out	Invalid

Constants / Literals

- Constants in C++ refer to **fixed values** that do not change during execution of program.



CONSTANTS

NUMERIC CONSTANTS

CHARACTER CONSTANTS

INTEGER CONSTANTS
i.e.
123,-321, 6543

REAL CONSTANTS
i.e.
0.0083, -0.75

SINGLE CHARACTER CONSTANTS
i.e.
'5', 'X', ';'

STRING CONSTANTS
i.e.
"Hello", "197"

C++ Operators

C++ Operators

- All C language operators are valid in C++.
    1. Arithmetic operators (+, - , *, /, %)
    2. Relational operators (<, <=, >, >=, ==, !=)
    3. Logical operators (&&, ||, !)
    4. Assignment operators (+=, -=, *=, /=)
    5. Increment and decrement operators (++, --)
    6. Conditional operators (?:)
    7. Bitwise operators (&, |, ^, <<, >>)
    8. Special operators ()

Arithmetic Operators

Operator	**example**	**Meaning**
+	a + b	Addition
-	a – b	Subtraction
*	a * b	Multiplication
/	a / b	Division
%	a % b	Modulo division- remainder

Relational Operators

Operator	**Meaning**
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Equal to
!=	Not equal to

Logical Operators

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

a	b	a && b	a || b
true	true		
true	false		
false	true		
false	false		

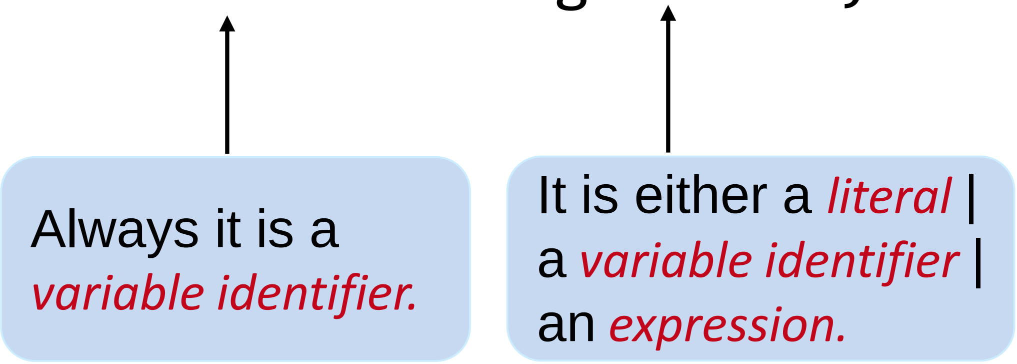
❑ a && b : returns false if any of the expression is false

❑ a || b : returns true if any of the expression is true

Assignment operator

- We assign a value to a variable using the basic assignment operator (=).
- Assignment operator stores a value in memory.
- The syntax is

```
leftSide = rightSide ;
```



Always it is a *variable identifier.*

It is either a *literal* | a *variable identifier* | an *expression.*

```
Literal: ex. i = 1;
Variable identifier: ex. start = i;
Expression: ex. sum = first + second;
```

Assignment Operators (Shorthand)

Syntax:

```
leftSide  Op= rightSide ;
```

It is an *arithmetic operator.*

```
Ex:
    x=x+3;
    x+=3;
```

Simple assignment operator	Shorthand operator
`a = a+1`	`a += 1`
`a = a-1`	`a -= 1`
`a = a * (m+n)`	`a *= m+n`
`a = a / (m+n)`	`a /= m+n`
`a = a % b`	`a %= b`

Increment and Decrement Operators

- **Increment ++**
  The ++ operator used to increase the value of the variable by one
- **Decrement – –**
  The – – operator used to decrease the value of the variable by one

Example:

```
        x=100;
        x++;
```

After the execution the value of x will be 101.

Example:

```
        x=100;
        x--;
```

After the execution the value of x will be 99.

Pre & Post Increment operator

Operator	Description
Pre increment operator (**++x**)	value of **x** is incremented before assigning it to the variable on the left

```
x = 10 ;
p = ++x;
```

First increment value of x by one

After execution
x will be **11**
p will be **11**

Operator	Description
Post increment operator (**x++**)	value of **x** is incremented after assigning it to the variable on the left

```
x = 10 ;
p = x++;
```

First assign value of x

After execution
x will be **11**
p will be **10**

What is the output of this program?

```
#include <iostream>
using namespace std;
int main ()
{
    int x, y;
    x = 5;
    y = ++x * ++x;
    cout << x << y;
    x = 5;
    y = x++ * ++x;
    cout << x << y;
}
```

(A) 749735

(B) 736749

(C) 367497

(D) none of the mentioned

Conditional Operator

Syntax:

exp1 ? exp2 : exp3

Working of the ? Operator:

- exp1 is evaluated first
  - if exp1 is true(nonzero) then
    - exp2 is evaluated and its value becomes the value of the expression
  - If exp1 is false(zero) then
    - exp3 is evaluated and its value becomes the value of the expression

```
Ex:
m=2;
n=3;
r=(m>n) ? m : n;
```

Value of **r** will be **3**

```
Ex:
m=2;
n=3;
r=(m<n) ? m : n;
```

Value of **r** will be **2**

Bitwise Operator

Operator	**Meaning**
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

Bitwise Operator Examples

```
8 = 1000 (In Binary)
6 = 0110 (In Binary)
```

Bitwise & (AND)

```
int a=8,b=6,c;
c = a & b;
cout<<"Output ="<< c;
```

```
Output = 0
```

Bitwise | (OR)

```
int a=8,b=6,c;
c = a | b;
cout<<"Output ="<< c;
```

```
Output = 14
```

Bitwise << (Shift Left)

```
int a=8,b=6,c;
c = a << 1;
cout<<"Output ="<< c;
```

```
Output = 16
```

left shifting is the equivalent of multiplying **a** by a power of two

Bitwise >> (Shift Right)

```
int a=8,b=6,c;
c = a >> 1;
cout<<"Output ="<< c;
```

```
Output = 4
```

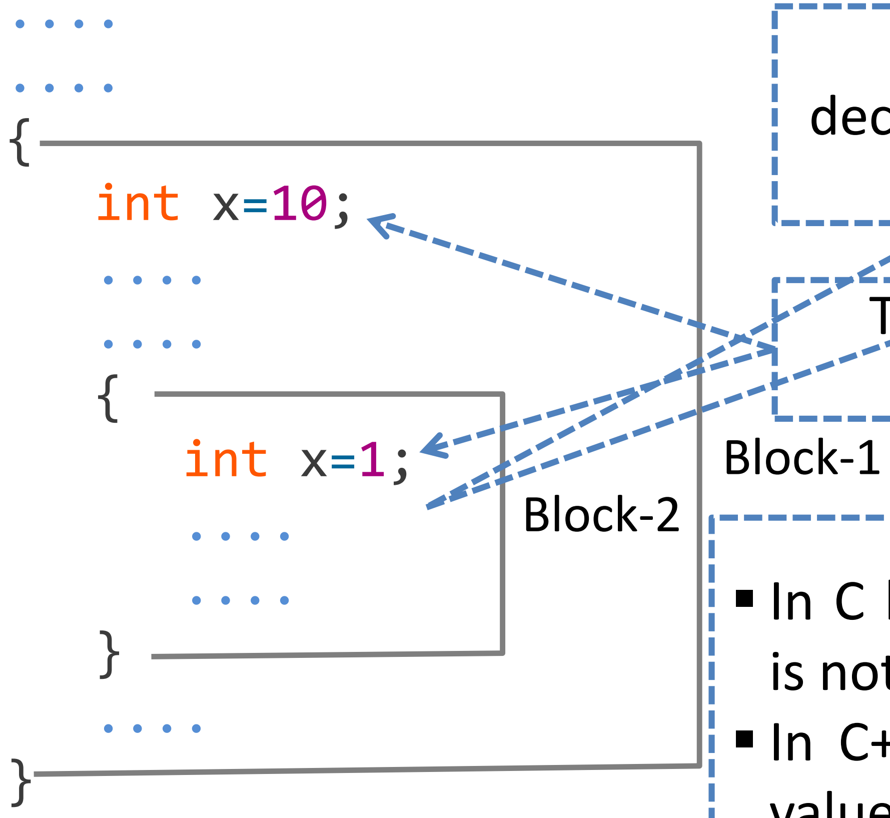
right shifting is the equivalent of dividing **a** by a power of two

New Operators in C++

`::`	Scope Resolution	It allows to access to the global version of variable
`::*`	Pointer-to-member declarator	Declares a pointer to a member of a class
`->*`	Pointer-to-member operator	To access pointer to class members
`.*`	Pointer-to-member operator	To access pointer to data members of class
`new`	Memory allocation operator	Allocates memory at run time
`delete`	Memory release operator	Deallocates memory at run time
`endl`	Line feed operator	It is a manipulator causes a linefeed to be inserted
`setw`	Field width operator	It is a manipulator specifies a field width for printing value

Scope Resolution Operator

Scope Resolution Operator(::)



```
....
....
{
    int x=10;
    ....
    ....
    {
        int x=1;
        ....
        ....
    }
    ....
}
```

Declaration of x in inner block hides declaration of same variable declared in an outer block.

Therefore, in this code both variable x refers to different data.

- In C language, value of x declared in Block-1 is not accessible in Block-2.
- In C++, using scope resolution operator (::), value of x declared in Block-1 can be accessed in Block-2.

Scope resolution example

```
#include <iostream>
using namespace std;
int m=10;
int main()
{
   int m=20;
   {
     int k=m;
     int m=3;
     cout<<"we are in inner block\n";
     cout<<"k="<<k<<endl;
     cout<<"m="<<m<<endl;
     cout<<"::m="<<::m<<endl;
   }
   cout<<"we are in outer block\n";
   cout<<"m="<<m<<endl;
   cout<<"::m="<<::m<<endl;
   return 0;
}
```

Global declaration of variable m

variable m declared , local to main

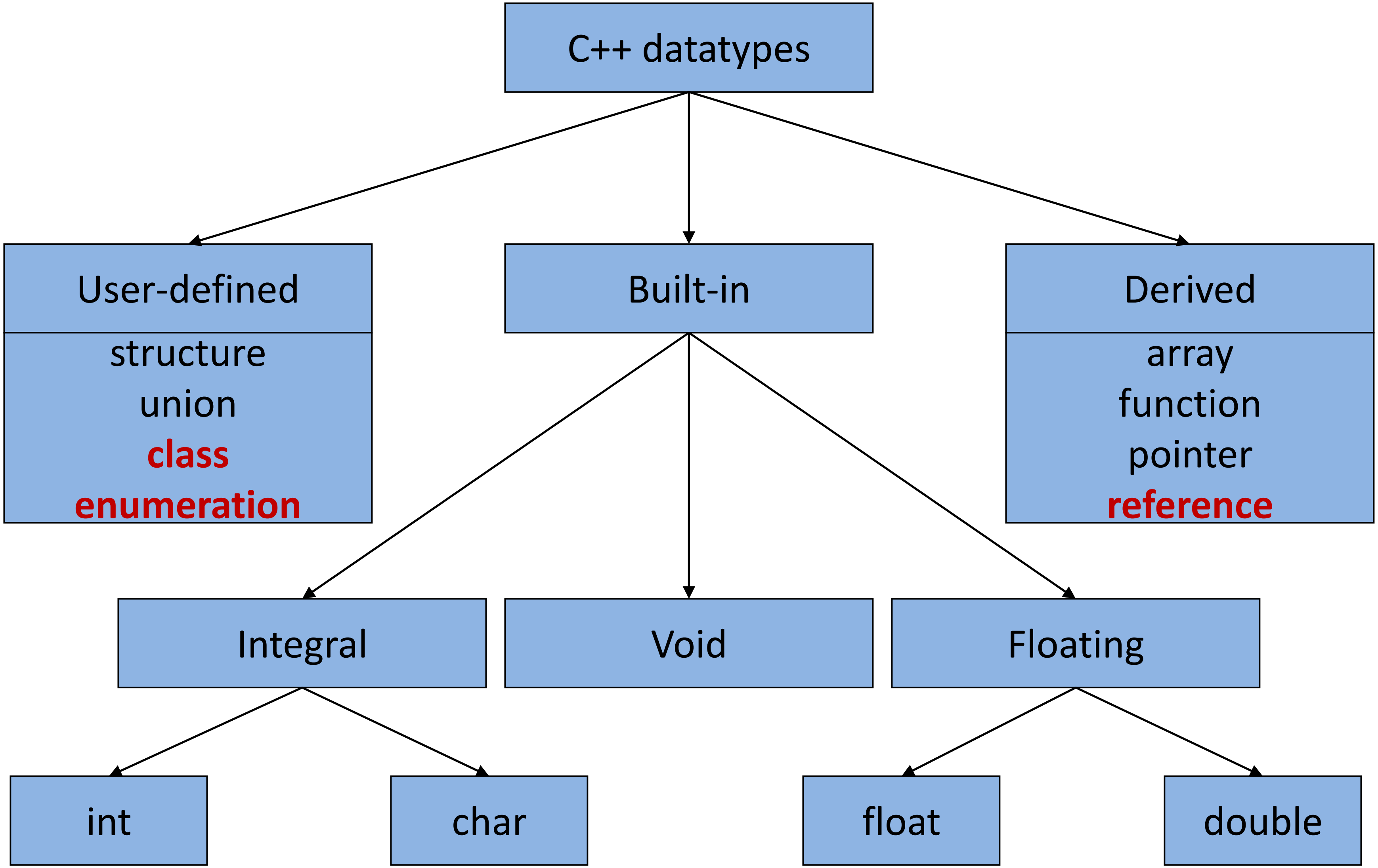
variable m
declared again local to inner block

Output:

```
we are in inner block
k=20
m=3
::m=10
we are in outer block
m=20
::m=10
```

C++ Data Types

Basic Data types



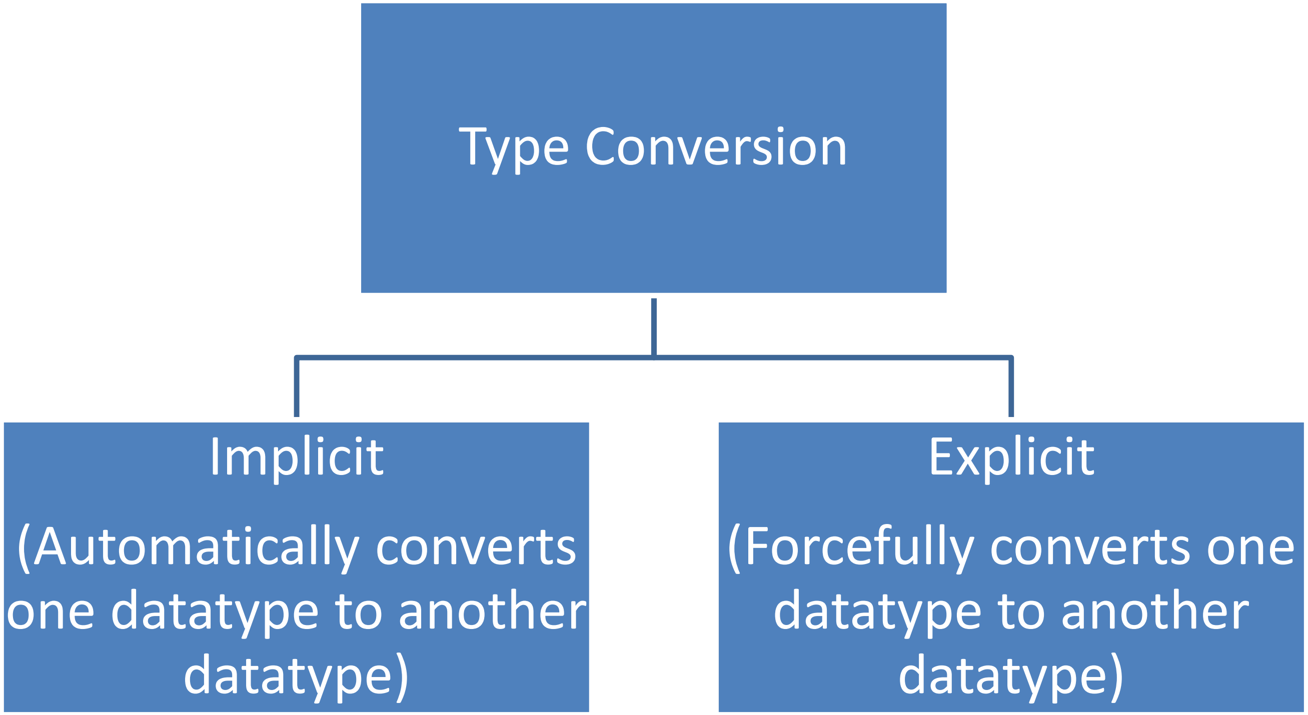
Built in Data types

Data Type	Size (bytes)	Range
char	1	-128 to 127
unsigned char	1	0 to 255
short or int	2	-32,768 to 32,767
unsigned int	2	0 to 65535
long	4	-2147483648 to 2147483647
unsigned long	4	0 to 4294967295
float	4	3.4e-38 to 3.4e+308
double	8	1.7e-308 to 1.7e+308
long double	10	3.4e-4932 to 1.1e+4932

Type Conversion

Type Conversion

- **Type Conversion** is the process of converting one predefined data type into another data type.



Type Conversion

Implicit

(Automatically converts one datatype to another datatype)

Explicit

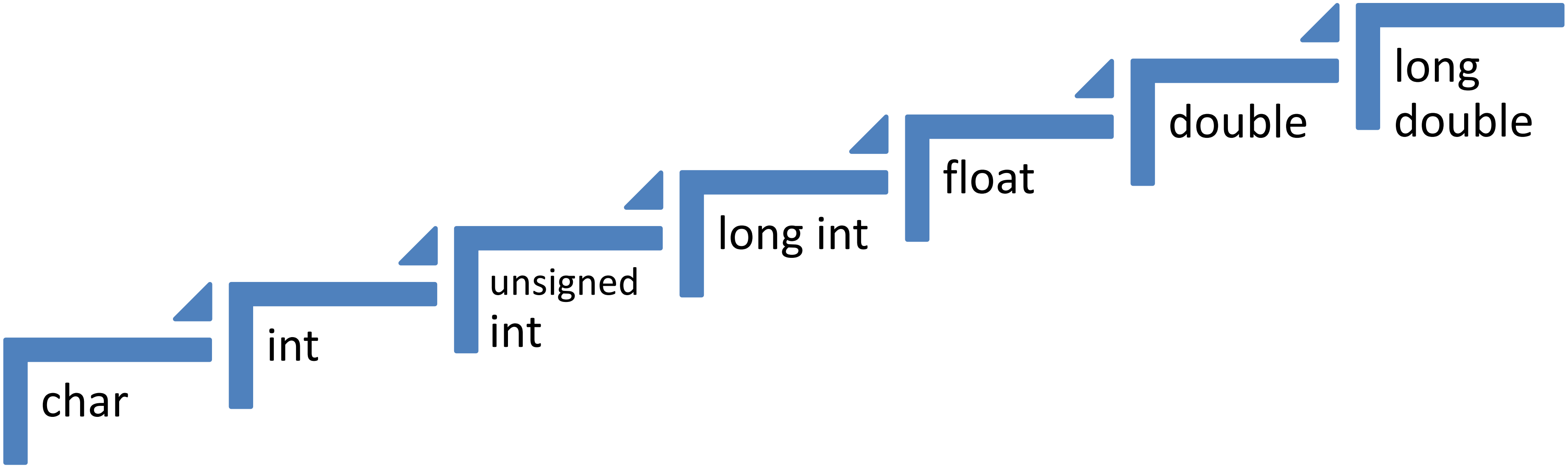
(Forcefully converts one datatype to another datatype)

- Explicit type conversion is also known as **type casting**.

Type Conversion(Cont…)

```
int a;

double b=2.55;

a = b; // implicit type conversion

cout << a << endl; // this will print 2

a = int(b); //explicit type conversion

cout << a << endl; // this will print 2
```

Implicit type conversion hierarchy



char

int

unsigned int

long int

float

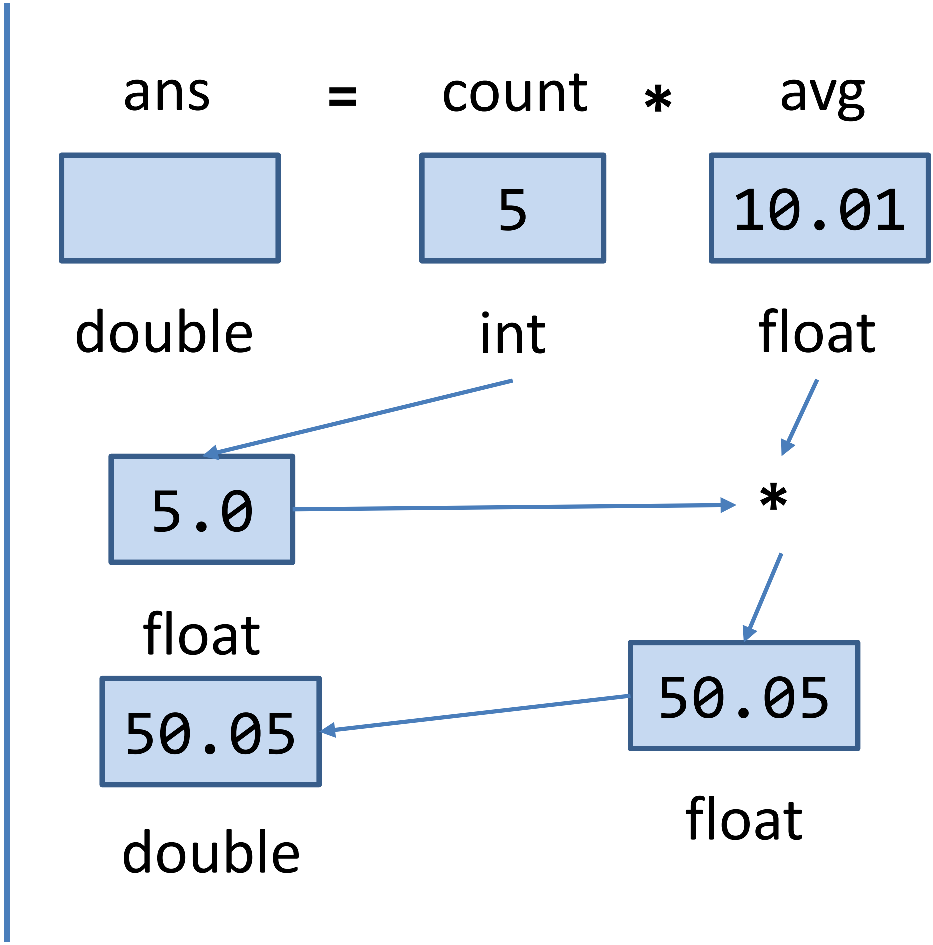
double

long double

Implicit Type Conversion

```
#include <iostream>
using namespace std;
int main()
{
    int count = 5;
    float avg = 10.01;
    double ans;

    ans = count * avg;

    cout<<"Answer=:"<<ans;
    return 0;
}
```

```
Output:
Answer = 50.05
```



Type Casting

- In C++ explicit type conversion is called **type casting.**
- Syntax

```
type-name (expression) //C++ notation
```

- Example

```
average = sum/(float) i; //C notation
average = sum/float (i); //C++ notation
```

Type Casting Example

```
#include <iostream>
using namespace std;
int main()
{
   int a, b, c;
   a = 19.99 + 11.99; //adds the values as float
                      // then converts the result to int
   b = (int) 19.99 + (int) 11.99;  // old C syntax
   c = int (19.99) + int (11.99); // new C++ syntax

   cout << "a = " << a << ", b = " << b;
   cout << ", c = " << c << endl;

   char ch = 'Z';
   cout << "The code for " << ch << " is ";  //print as char
   cout << int(ch) << endl;   //print as int
   return 0;
}
```

Output:

```
a = 31, b = 30, c = 30
The code for Z is 90
```

Reference Variable

Reference Variable

- A **reference** provides an alias or a different name for a variable.
- One of the most important uses for references is in passing arguments to functions.

```
int a=5;
int &ans = a;

cout<<"a="<<a<<endl;
cout<<"&a="<<&a<<endl;
cout<<"ans="<<ans<<endl;
cout<<"&ans="<<&ans<<endl;
ans++;
cout<<"a="<<a<<endl;
cout<<"ans="<<ans<<endl;
```

declares variable a

declares ans as reference to a

OUTPUT

```
a=5
&a=0x6ffe34
ans=5
&ans=0x6ffe34

a=6
ans=6
```

Its necessary to initialize the Reference at the time of declaration

Reference Variable(Cont…)

- C++ references allow you to create a second name for the a variable.
- **Reference variable** for the purpose of accessing and modifying the value of the **original variable** even if the second name (the reference) is located within a **different scope**.

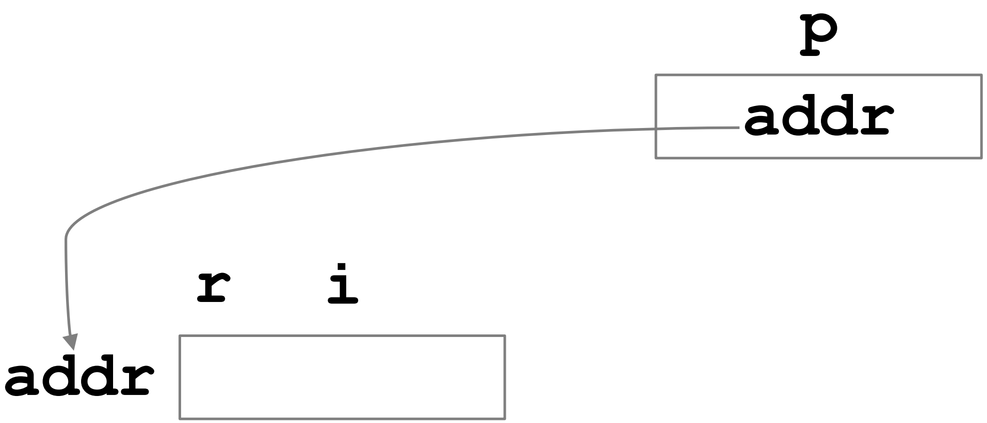
Reference Vs Pointer

References

```
int i;
int &r = i;
```

Pointers

```
int *p = &i;
```



- A reference is a variable which **refers** to another variable.

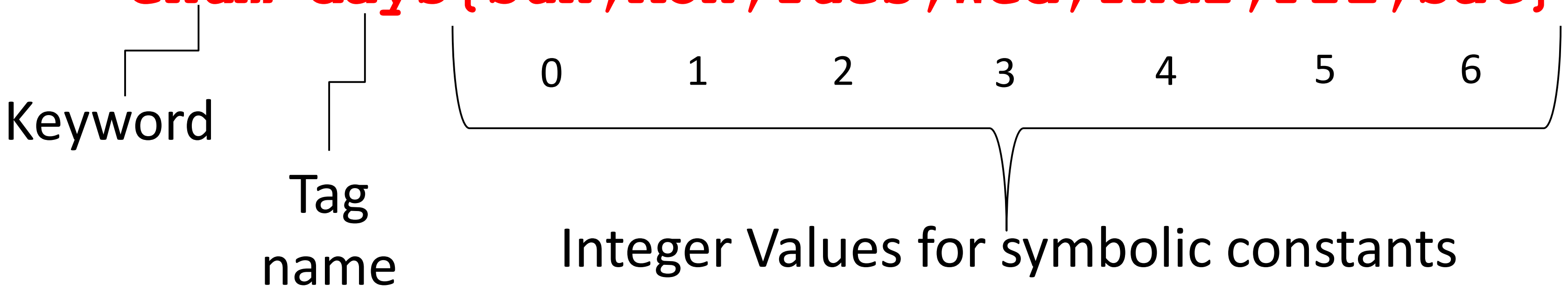
- A pointer is a variable which **stores the address** of another variable.

Enumeration

Enumeration (A user defined Data Type)

- An **enumeration** is set of named **integer** constants.
- Enumerations are defined much like structures.

```
enum days{Sun,Mon,Tues,Wed,Thur,Fri,Sat};
```



- Above statement creates **`days`** the name of datatype.
- By default, enumerators are assigned <u>integer values starting with 0</u>.
- It establishes **`Sun`**, **`Mon`**… and so on as symbolic constants for the integer values 0-6.

Enumeration Behaviour(Cont…)

```
enum coin { penny, nickel, dime, quarter=100,
            half_dollar, dollar};
```

The values of these symbols are

penny	0
nickel	1
dime	2
quarter	100
half_dollar	101
dollar	102

Enumeration Behaviour

```
enum days{ sun, mon, tue, wed, thu, fri, sat };
days today;
```

variable **today** declared of type **days**

```
today = tue;
```

Valid, because tue is an enumerator. Value 2 will be assigned in today

```
today = 6;
```

Invalid, because 6 is not an enumerator

```
today++;
```

Invalid, today is of type days. We can not apply ++ to structure variable also

```
today = mon + fri;
```

Invalid

```
int num = sat;
```

Valid, days data type converted to int, value 6 will be assigned to num

```
num = 5 + mon;
```

Valid, mon converted to int with value 1

Control Structures

Control Structures

- The **`if`** statement:
    - Simple **`if`** statement
    - **`if…else`** statement
    - **`else…if`** ladder
    - **`if…else`** nested
- The **`switch`** statement :
- The **`do-while`** statement: An exit controlled loop
- The **`while`** Statement: An entry controlled loop
- The **`for`** statement: An entry controlled loop

FUNCTIONS IN C++

Introduction

- Dividing a program into functions.
  - a major principle of top-down, structured programming.
- To reduce the size of the program.
- Code re-use.
- C++ function can be overloaded to make it perform different tasks depending on the arguments passed to it.

Introduction

continue…

Two types of Function:

Library Function : getch(),clrscr()

User Defined Function

```
void show( );    /* Function declaration OR Function Prototype*/
void main( )
{
  ----
  show( );       /* Function call */
  ----
}
void show( )     /* Function definition */
{
  ----
  ----           /* Function body */
}
```

The main( ) Function

- The main( ) returns a value of type int to the operating system by default.
- The functions that have a return value should use the return statement for termination.
- Use void main( ), if the function is not returning any value.

Function Prototyping

- The prototype describes the function interface to the compiler by giving details such as:
  - The number and type of arguments
  - The type of return values.
- It is a template
- When the function is called, the compiler uses the template to ensure that proper arguments are passed, and the return value is treated correctly.

Function Prototyping

continue…

- Function prototype is a declaration statement in the calling program.

type function-name (argument-list) ;

- The argument-list contains the types and names of arguments that must be passed to the function.

Function Prototyping

continue…

- Each argument variable must be declared independently inside the parentheses.

```
 float avg ( int x, int y) ;     //  correct
float avg ( int x, y) ;          //  illegal
```

- In a function declaration, the names of the arguments are dummy variables and therefore they are optional.

Function Prototyping

continue…

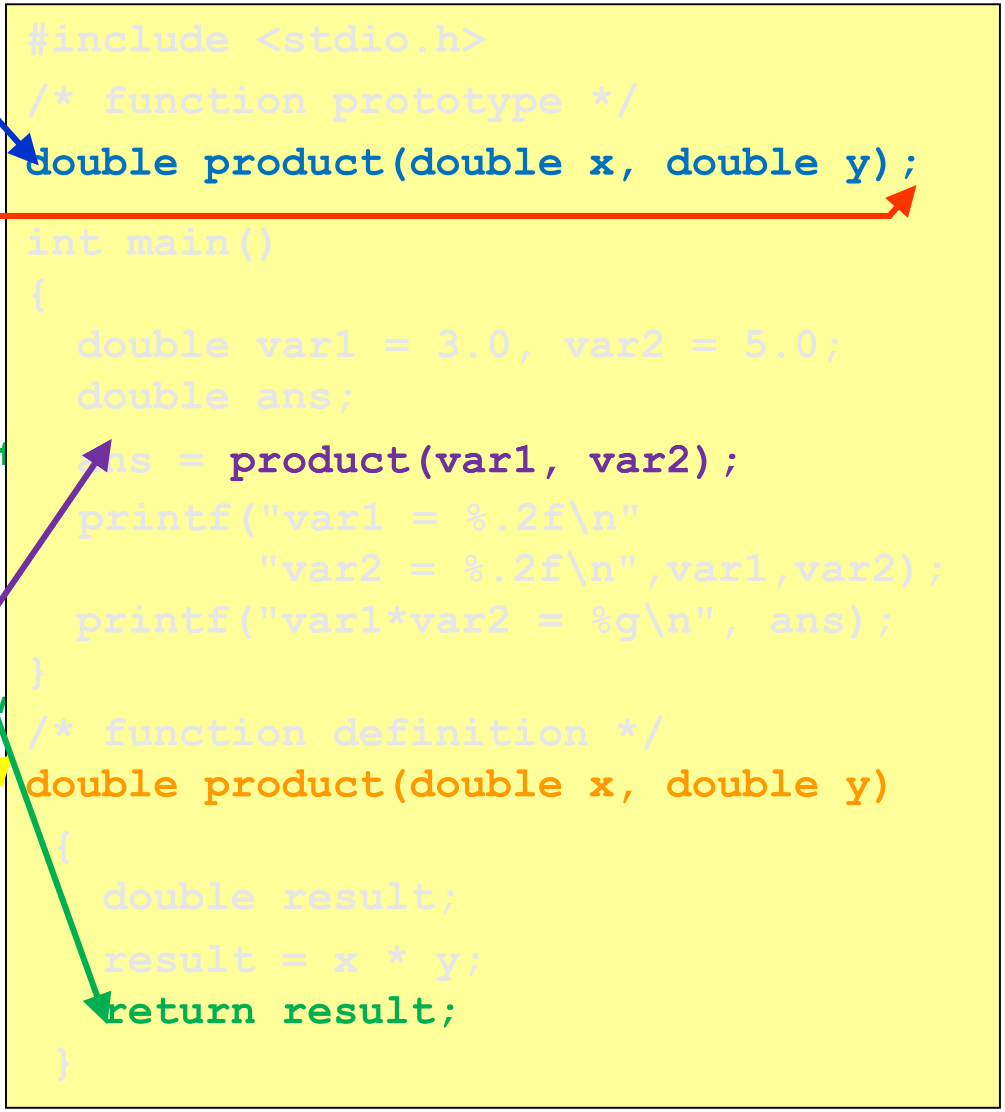
```
float avg ( int , int ) ;
```

The variable names in the prototype just act as placeholders and therefore if names are used, they do not have to match the names used in the ***function call*** or ***function definition***.

```
void display( );          // function with an
void display(void);     // empty argument list.
```

Functions - Example

- **Function prototype**
  - **Like a variable declaration**
    - **Tells compiler that the function will be defined later**
    - **Helps detect program errors**
    - **Note semicolon!!**
- **Function definition**
  - **Note, semicolon**
- **Function return**
  - **return statement terminates execution of the current function**
  - **Control returns to the calling function**
  - **if return expression;**
    - **then value of expression is returned as the value of the function call**
    - **Only one value can be returned this w**
- **Function call**
  - **main() is the 'calling function'**
  - **product() is the 'called function'**
  - **Control transferred to the function code**
  - **Code in function definition is executed**



```
#include <stdio.h>
/* function prototype */
double product(double x, double y);
int main()
{
  double var1 = 3.0, var2 = 5.0;
  double ans;
  ans = product(var1, var2);
  printf("var1 = %.2f\n"
         "var2 = %.2f\n",var1,var2);
  printf("var1*var2 = %g\n", ans);
}
/* function definition */
double product(double x, double y)
 {
   double result;
   result = x * y;
   return result;
 }
```

Types of User-defined Functions in C Programming

- For better understanding of arguments and return in functions, user-defined functions can be categorised as:
- Function with no arguments and no return value
- Function with no arguments and return value
- Function with arguments but no return value
- Function with arguments and return value.

Sr. No	C function	Syntax

1	with arguments and with return values	int function (int); // function declaration
		function (a); // function call
		int function(int a) // function definition
		{statements;
		 return a;}

2	with arguments and without return values	void function (int); // function declaration
		function(a); // function call
		void function(int a) // function definition
		{statements;}

3	without arguments and without
	return values
		void function(); // function declaration
		function(); // function call
		void function() // function definition
		{statements;}

4	without arguments and with return values	int function ( ); // function declaration
		function ( ); // function call
		int function( ) // function definition
		{statements;
		return a;}

Function with no arguments and no return value

```
#include<stdio.h>
#include<conio.h>
void area();
void main()
{
area();
getch();
}
void area()
{
        float r, ar;
        printf("enter radius of circle");
        scanf("%f",&r);
        ar=3.14*r*r;
        printf("the area of circle is %f",ar);
}
```

Function with no arguments and no return value

Function with Arguments but not Return Value

```
#include<stdio.h>
#include<conio.h>
void area(float r);
void main()
{
        float x;
        printf("enter radius of circle");
        scanf("%f",&x);
area(x);
getch();

}
void area(float r)
{
        float  ar;

        ar=3.14*r*r;
        printf("the area of circle is %f",ar);
}
```

Function with arguments but no return value

Function with Arguments and Return Value

```
#include<stdio.h>
#include<conio.h>
float area(float r);
int main()
{
        float x,ans;
        printf("enter radius of circle");
        scanf("%f",&x);
ans=area(x);
printf("%f the area of a circle is",ans);
return 0;       }
}
float area(float r)
{
        float  ar;
        ar=3.14*r*r;
     return(ar);
}
```

Function with arguments and return value.

Function with no arguments and return value

```
#include<stdio.h>
#include<conio.h>
float area();
int main()
{ float ans;

ans=area();
printf("%f the area of a circle is",ans);
return 0;
}
float area()
{
          float x,ar;
          printf("enter radius of circle");
          scanf("%f",&x);


          ar=3.14*x*x;
     return(ar);
}
```

Function with no arguments and return value

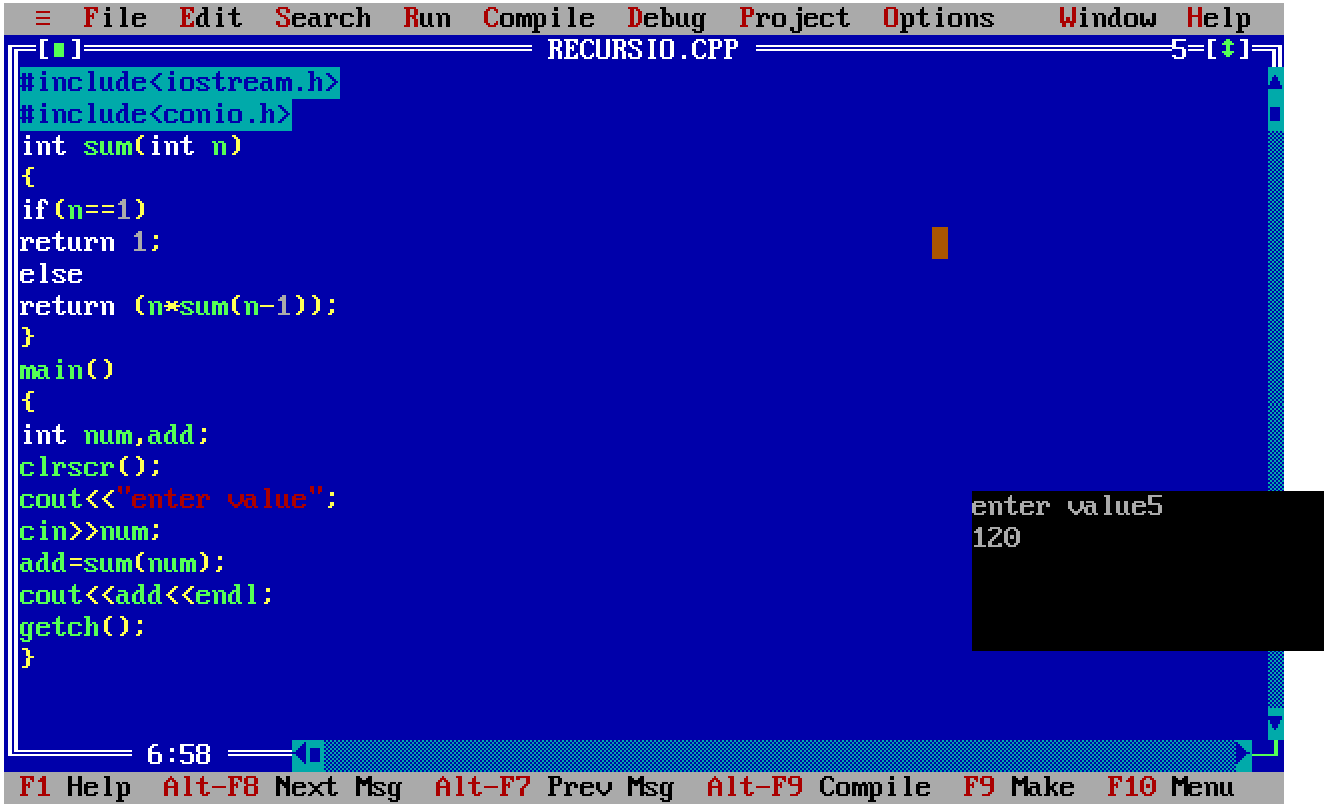
C++ Programming Recursion

- A function that calls itself is known as recursive function and the process of calling function itself is known as recursion in C++ programming.
- Example of recursion in C++ programming
- **Write a C++ program to find sum of first n natural numbers using recursion. Note: Positive integers are known as natural number i.e. 1, 2, 3....n**

```
#include <iostream>
using namespace std;
int sum(int n);
int main()
{
    int num,add;
    cout<<"Enter a positive integer:\n";
    cin>>num;
    add=sum(num);
    cout<<add;
}
int sum(int n){
    if(n==0)
       return n;
    else
       return n+sum(n-1); /*self call  to function sum() */
}
```

- sum(5)
- =5+sum(4)
- =5+4+sum(3)
- =5+4+3+sum(2)
- =5+4+3+2+sum(1)
- =5+4+3+2+1+sum(0)
- =5+4+3+2+1+0
- =5+4+3+2+1
- =5+4+3+3
- =5+4+6
- =5+10
- =15
- Every recursive function must be provided with a way to end the recursion. In this example when, n is equal to 0, there is no recursive call and recursion ends.

Recursion of Factorial Number



≡ File Edit Search Run Compile Debug Project Options Window Help
[■] RECURSIO.CPP 5=[↕]
6:58
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
#include<iostream.h>
#include<conio.h>
int sum(int n)
{
if(n==1)
return 1;
else
return (n*sum(n-1));
}
main()
{
int num,add;
clrscr();
cout<<"enter value";
cin>>num;
add=sum(num);
cout<<add<<endl;
getch();
}
```

```
enter value5
120
```

Call by Value

A function call passes arguments by value.

- The called function creates a new set of variables and copies the values of arguments into them.

- The function does not have access to the actual variables in the calling program and can only work on the copies of values.

```
#include<iostream>
//#include<conio.h>
uisng namespace std;
int  swap(int  ,int  );
int main()
{       int a,b;
cout<<“enter a,b”;
cin>>a>>b;
swap(a,b);
return 0;
}
```

```
• int swap(int a,int b)
• {
•    int t=a;
•    a=b;
•    b=t;
•  cout<<a<<“    “<<b;
• }
```

Call by Reference

When we pass arguments by reference, the formal arguments in the called function become aliases to the actual arguments in the calling function.

This means that when the function is working with its own arguments, it is actually working on the original data.

```
#include<iostream>
using namespace std;
int swap(int  &,int  &);
int main()
{
  int a,b;
cout<<“enter a,b”;
cin>>a>>b;
swap(a,b);
cout<<a<<b;
return 0;
}
```

```
int swap(int  &a,int &b)
{
  int t=a;
  a=b;
  b=t;
}
```

Inline Function

- We mentioned that functions save memory space because all the calls to the function cause the same code to be executed
- The function body need not be duplicated in memory
- When compiler sees the function call, it normally generates a jump to the function.
- At end of the function it jumps back to the instruction following the call.
- While this sequence of events may save memory space, it takes some extra time.

- **Syntax:-** inline function-header
    - {
-         function body
    - }

- **Eg:** inline double cube(double a)
- {
- return (a*a*a);
- }

Max. from 3 no. using Inline Function

- #include<iostream.h>
- #include<conio.h>
- inline int max(int a,int b,int c)
- {   return (a>b ?((a>c)?a:c) : ((b>c? b: c)));   }
- int main()
- {   int x,y,z;
- cout<<“Enter x,y,z”;
- cin>>x>>y>>z;
- cout<< “Max no is”<<max(x,y,z);   getch();

```
Enter x,y,z
1      2     3

Max no is3
```

Function Overloading

- **Same thing for different purposes.**

```
#include<iostream.h>
#include<conio.h>
void add(int ,int );
void add(float ,int );
void add(float ,float );
void main()
{
int a,b;
float c,d;
cout<<“Enter a,b,c,d”;
cin>>a>>b>>c>>d;
add(a,b);
add(c,a);
add(c,d);
getch();
}
void add(int a,int b)
{
 cout<<a+b;  }
void add(float c,int a)
{ cout<<c*a;   }
void add(float c,float  d)
{cout<<c/d;    }
```

Storage Class Specifier

- It provides information about variable’s visibility and lifetime.
- It can be declared as……

- Auto
- Register
- Static
- External

Auto Storage Class

- Local variables which are declared inside a function.
- Memory space for local variable is automatically allocated when function is executing and released as soon as it ends.
- Section of program where variable can be used is called scope of variable. Local variables are given the **storage class auto by default.**
- **Example:** int main()   { auto int a,b; return 0;}
  - **OR**
- int main() { int a,b; return 0;}

Example

```
#include<iostream.h>
#include<conio.h>
void test();
int main()
{  int a=5;
cout << a<<endl;
{  int a=10;     //print 10
cout<<a<<endl;  }
test();
return 0;
}
```

```
• void test()
• {
• int b=6;    // local variable to test()
• cout<<b<<endl;
• getch();
• }
```

Output:

```
5 10 6
```

Register Storage Class

- Local variables stored in register instead of memory as register is much faster than memory.
- Variable can be declared by register.
- register int a;
- It is just a request.
- It depends on compiler to store variable in register or not.

Static Storage Class

- he static variable is initialized only once and exists till the end of a program. It retains its value between multiple functions call.

- The static variable has the default value 0 which is provided by compiler.

- For defining static variable use **static as keyword.**

```
#include <iostream>
using namespace std;
void func() {
   static int i=0; //static variable
   int j=0; //local variable
   i++;
   j++;
   cout<<"i=" << i<<" and j=" <<j<<endl;
}
int main()
{
 func();
 func();
 func();
}
```

Extern Storage Class

- The extern storage class specifier allows you to declare variables and functions that several source file can use.

- It gives reference of global variable that is visible to all the program files.

- Extern specifier is mostly used when there are two or more files sharing same global variables or functions.

- extern int globalvar;   // declare that variable is defined in another file (Extern variable)

- int globalvar=28;   //Definition

  //OR

- extern int globalvar=28;   //Extern” specifier is optional.

File: sub.cpp

```
int test=100;

void multiply(int n)

{

    test=test*n;

}
```

File: main.cpp

```
#include<iostream>

#include "sub.cpp"  // includes the content of sub.cpp

using namespace std;

extern int test;  // declaring test


int main()

{

    cout<<test<<endl;

    multiply(5);

    cout<<test<<endl;

    return 0;

}
```

Manipulators

Manipulators

- endl Manipulator
- setw Manipulator
- setfill Manipulator
- setprecision Manipulaton

Endl Manipulator

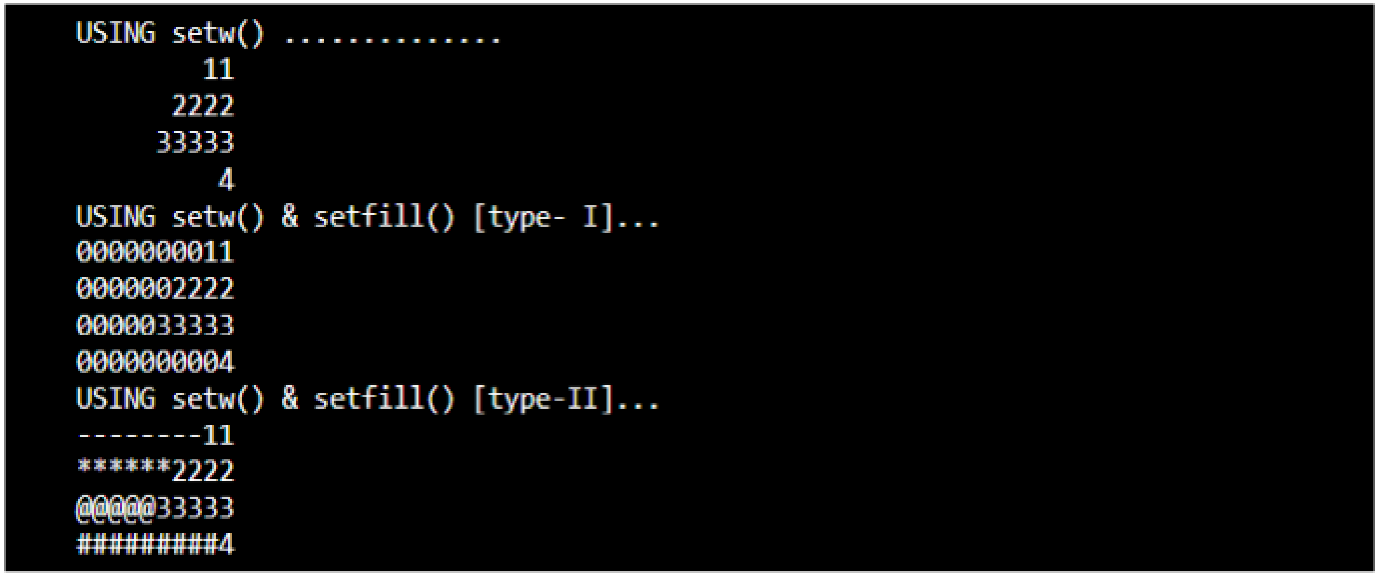
```
#include <iostream.h>

#include<conio.h>

void main()

{

        cout<<"USING '\\n' ...\n";

        cout<<"Line 1 \nLine 2 \nLine 3 \n";

        cout<<"USING end ..."<< endl;

        cout<< "Line 1" << endl << "Line 2" << endl << "Line 3" << endl;

getch();

}
```

setw and setfill Manipulator

- setw manipulator sets the width of the filed assigned for the output.
- The field width determines the minimum number of characters to be written in some output representations.
- setfill character is used in output insertion operations to fill spaces with characters when standard width of the representation is shorter than the field width.

setw and setfill Manipulator

```
#include <iomanip.h>

 int main()

 {           cout<<"USING setw() ..............\n";

             cout<< setw(10) <<11<<"\n";

             cout<< setw(10) <<2222<<"\n";

             cout<< setw(10) <<33333<<"\n";

             cout<< setw(10) <<4<<"\n";

cout<<"USING setw() & setfill() [type- I]...\n";

             cout<< setfill('0');

             cout<< setw(10) <<11<<"\n";

cout<< setw(10) <<2222<<"\n";

        cout<< setw(10) <<33333<<"\n";

        cout<< setw(10) <<4<<"\n";

      cout<<"USING setw() & setfill() [type-II]...\n";

        cout<< setfill('-')<< setw(10) <<11<<"\n";

        cout<< setfill('*')<< setw(10) <<2222<<"\n";

        cout<< setfill('@')<< setw(10) <<33333<<"\n";

        cout<< setfill('#')<< setw(10) <<4<<"\n";

        return 0;

•    }
```



```
USING setw() ..............
        11
      2222
     33333
         4
USING setw() & setfill() [type- I]...
0000000011
0000002222
0000033333
0000000004
USING setw() & setfill() [type-II]...
--------11
******2222
@@@@@33333
#########4
```

Setprecision manipulator

- setprecision manipulator sets the total number of digits to be displayed, when floating point numbers are printed.
- Syntax: setprecision([number_of_digits]);
- Example:

```
cout<<setprecision(5)<<1234.537;

// output will be : 1234.5
```

Previous Exam Questions

- What is function? Demonstrate types of function with their syntax.
- What is inline function? Explain with suitable example.
- Difference between call by value and call by reference.
- Explain call by value and call by reference with suitable example (swap program)
- Explain default argument with suitable example
- Difference between macro and inline function.
- **C-2-mefgi@googlegroups.com**

Thank You